SERVIÇO PÚBLICO FEDERAL

UNIVERSIDADE FEDERAL DE SERGIPE

PRÓ-REITORIA DE PÓS-GRADUAÇÃO E PESQUISA

PROGRAMA INSTITUCIONAL DE BOLSAS DE INICIAÇÃO CIENTÍFICA – PIBIC

Explorando Algoritmos Baseados em Enxames de Partículas No Contexto do Problema do Escalonamento e Atribuição de Tarefas em Projetos de Software

Relatório Final

Área do conhecimento: Engenharia de Software

Subárea do conhecimento: Otimização em Engenharia de Software

Especialidade do conhecimento: Hiper-heurísticas

Este projeto é desenvolvido com bolsa de iniciação científica

PIBIC/COPES

Orientador: Leila Maciel de Almeida e Silva

Autor: José Joaquim de Andrade Neto

Resumo

Escalonar funcionários em um projeto de software pode ser uma tarefa complexa de ser realizada manualmente e, com isso, acarretar acréscimos na duração e no custo do projeto. Dessa forma, são usados algoritmos de otimização para gerar soluções automáticas e que minimizem esses problemas. No entanto, definir um algoritmo e configurar os seus parâmetros é uma tarefa dispendiosa, exigindo, em sua maioria, testes empíricos para a validação daquela configuração. Hiper-heurísticas são algoritmos que geram outras heurísticas e suas configurações, tornando-se uma forma mais fácil de resolver um dado problema. Esse projeto estudou a aplicação de hiper-heurística para gerar configurações de parametrização para a meta-heurística de Otimização por Enxame de Partículas, na tentativa de melhorar os resultados já vistos na literatura para o problema do escalonamento. A análise dos resultados sugerem que os algoritmos gerados obtiveram uma performance melhor do que as já existentes na literatura.

Palavras-chaves: SPSP. PSO. hiper-heurística. meta-heurística. planejamento de projeto de software. otimização combinatória.

Lista de ilustrações

Lista de Gramáticas

Lista de tabelas

Lista de abreviaturas e siglas

BNF	Backus-Naur Form
CRO	Chemical Reaction Optimization
GA	Genetic Algorithm
KE	Kinect Energy
MD	Matriz de Dedicação
NCRO	Nondominated Sorting Chemical Reaction Optimization
NSGA-II	Nondominated Sorting Genetic Algorithm II
PSO	Particle Swarm Optimization
SBSE	Search Based Software Engineering
SMPSO	Speed-constrained Multi-Objective Particle Swarm Optimization
SPSP	Software Project Scheduling Problem
GDP	Grafo de Precedência de Tarefas

Sumário

1 Introdução

O Problema do Escalonamento em Projeto de Software - *Software Project Scheduling Problem* (SPSP) - é uma atividade que integra o ciclo de desenvolvimento de um software, fazendo parte de seu planejamento (SOMMERVILLE, 2015). Essa atividade, restrita ao líder do projeto em sua maioria (PRESSMAN, 2014), possui alta importância, uma vez que estatísticas apontam que somente 29% de 50.000 projetos do ano de 2015 obtiveram sucesso (GROUP, 2015), onde boa parte dos insucessos derivaram de falhas que aconteceram no planejamento. Dentre os considerados grandes, a situação ainda deixa mais evidente a importância de um bom planejamento, visto que apenas 2% dos projetos considerados grandes conseguiram chegar ao final de forma satisfatória.

Várias variantes do SPSP são estudadas na literatura. Uma das primeiras tentativas de modelagem remete-se ao *Resource-Constrained Software Project Scheduling* (RCSPS) (BRUCKERA et al., 1999). Nesse modelo, existem as atividades com suas durações, e existem os recursos com suas respectivas capacidades. Cada atividade se relaciona com alguma outra, indicando uma relação de pré-requisito. Ao iniciar uma atividade, a mesma consome uma quantidade específica de recursos, os quais são liberados ao seu término. O objetivo do RCSPS é alocar recursos para as tarefas de uma forma que todas sejam completadas da forma mais rápida, isto é, que o instante em que a última tarefa acaba seja o menor possível.

Recentemente, uma das modelagens mais utilizadas pela literatura do SPSP foi proposta em Alba e Chicano (2007). Diversos trabalhos na literatura relacionada buscaram novas investigações acerca do SPSP sob a ótica dessa modelagem (CHANG et al., 2008), (CHEN; ZHANG, 2013a), (XIAO; AO; TANG, 2013). Em linhas gerais, o líder do projeto deve escolher funcionários para desempenhar várias tarefas. Cada funcionário possui características individuais, como salário e habilidades, da mesma forma que cada tarefa possui um custo, habilidades necessárias para a sua execução, e relacionam-se entre si.

Contudo, todas as modelagens do SPSP são, de forma instríseca, difíceis, no que tange à sua complexidade computacional. Isso acontece, pois, todas essas modelagens pertencem a classe $\mathcal{NP}$-Difícil (CRAWFORD et al., 2014), e portanto, se desconhece a existência de um algoritmo exato que possa resolver estas variantes de forma eficiente. Dessa forma, os trabalhos da literatura optam por estudar o SPSP usando heurísticas (PEIXOTO; MATEUS; RESENDE, 2014), uma vez que as mesmas apresentam bons resultados (GOLDBARG; GOLDBARG; LUNA, 2016), ainda que estes não sejam ótimos.

Essas propostas estão englobadas pela área de Engenharia de Software Baseada em Busca - *Search-Based Software Engineering* (SBSE) (HARMAN; JONES, 2001) - uma área emergente que visa a solução de problemas de natureza difícil da Engenharia de Software,

através do uso, em sua maioria, de heurísticas e de métricas avaliadoras, capazes de quantificar o quão boa é uma solução. O poder de abstração das meta-heurísticas, que as fazem serem aplicadas de forma quase que independente do problema, é uma das características que impulsionou-as como o método mais utilizado na SBSE (HARMAN, 2007).

No entanto, mesmo que meta-heurísticas se abstenham do domínio do problema, elas ainda necessitam de configurações a partir de parâmetros. A tarefa de escolha desses parâmetros pode ser dispendiosa e até mesmo difícil (SABAR et al., 2013), em certos casos. Nesse contexto, as hiper-heurísticas aparecem como uma maneira para a geração de meta-heurísticas genéricas o suficiente. Elas são definidas como um método de busca ou mecanismo de aprendizagem para selecionar ou gerar heurísticas para resolverem problemas de buscas. Em outras palavras, hiper-heurísticas buscam boas meta-heurísticas ao invés de boas soluções. Ao adicionar mais um nível de abstração, hiper-heurísticas devem ser mais genéricas ainda para que consigam produzir soluções aceitáveis para diferentes instâncias de um mesmo problema (BURKE et al., 2013).

Finalmente, hiper-heurísticas ainda estão começando a ganhar corpo no contexto da SBSE. No entanto, os resultados que já existem na literatura são promissores. Dentre alguns, hiper-heurísticas foram usadas para escolhas de refatoração em software (MARIANI, 2017), testes e integridade de um software (JIA et al., 2015), para gerar algoritmos de busca (MARIANI et al., 2016), e para selecionar operadores de buscas a serem usados durante a execução de um algoritmo (GUIZZO et al., 2015), (JIA, 2015). Porém, hiper-heurísticas ainda não foram aplicadas em nenhum modelo de escalonamento de funcionários, sobretudo no SPSP. Dessa forma, cabe a investigação sobre o desempenho da meta-heurística de enxame de partículas (PSO) no contexto do especificado, na tentativa de aprimorar os resultados existentes.

1.1 Hipóteses

A hipótese principal desse projeto de pesquisa é de que, através da construção de uma gramática, que cubra um leque considerável de combinações para a escolha de parâmetros, e a escolha de uma hiper-heurística, é possível encontrar soluções para o SPSP melhores que as já existentes na literatura.

1.2 Objetivos do Projeto

O objetivo principal desse projeto é alcançar resultados melhores ou tão bom quantos os já existentes na literatura para o o SPSP. Para cumpri-lo, deve-se, também, a elaboração de uma hiper-heurística capaz de gerar novas meta-heurísticas tais quais o PSO, já englobando a criação de uma gramática que servirá de uso na determinação dos parâmetros do PSO.

1.3 Organização

Este presente relatório está organizado da seguinte forma:

Capítulo 2 - Revisão da Literatura: Esse capítulo apresenta todas definições e conceitos necessário para o entendimento desse projeto de pesquisa. Tais conceitos abordam problemas com muitos objetivos, o SPSP, as meta-heurísticas e as hiper-heurísticas;

Capítulo 3 - Metodologia: Esse capítulo detalha a metodologia seguida desde o início do projeto, bem como as atividades realizada. Além disso, também detalha o cronograma seguido durante a primeira parte desse projeto de pesquisa;

Capítulo 4 - Experimentos: Neste capítulo estão detalhadas todas as esquematizações, métricas, gramática e instância utilizada neste projeto. Todo o embasamento teórico foi usado para a definição dos instrumentos aqui definidos;

Capítulo 5 - Resultados: Explicitam a saída gerada dos experimentos, discutindo os por quês do que foi obtido e argumentando acerca do desempenho da hiper-heurística;

Capítulo 6 - Conclusões: Esse capítulo contempla as conclusões alcançadas através de todos os conceitos apresentados no Capítulo 2 em conjunto com o que foi apresentado nos resultados.

2 Revisão da Literatura

Para a investigação dos objetivos apresentados na seção 1.2, é preciso estudar e definir diversos conceitos que tangem o tema do projeto de pesquisa. Dessa forma, serão introduzidos na seção 2.1 os problemas com muitos objetivos, uma vez que o SPSP, formalizado na seção 2.2, é classificado como um problema desse tipo. Para a busca das soluções, será discutida a Engenharia de Software Baseada em Busca, na seção 2.3. Finalmente, as técnicas estudadas para a obtenção das soluções, as meta-heurísticas e hiper-heurísticas, são apresentadas na seções 2.4 e 2.7, respectivamente.

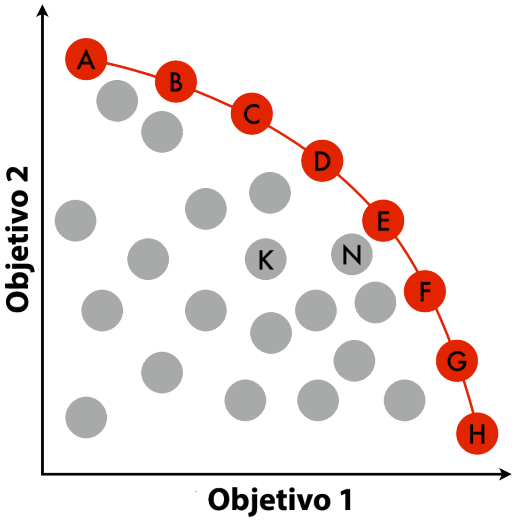
2.1 Problemas com Muitos Objetivos

Um problema é dito ser multiobjetivo (ou ter muitos objetivos) quando ele possui uma quantidade k de objetivos a serem otimizados simultaneamente, onde k é um número natural maior que 1. Formalmente, uma otimização com vários objetivos é composta por vetores de decisões $x = [x_1, x_2, ..., x_n]^T$, onde n são as variáveis de decisões e T é a transposição do vetor coluna - a representação usual - para o vetor linha, restrições de modelo limitações físicas ou temporais, por exemplo que são representadas por inequações $g_i \leq 0, i = \{1, 2, ..., m\}$ ou igualdades $h_j(x) = 0, j = \{1, 2, ..., p\}$ onde m e p representam a quantidade de restrições. Para a medição da qualidade de uma solução, usa-se funções objetivos, expressas a partir dos vetores de decisões. Uma função objetivo é definida como $f(x) = [f_1(x), ..., f_z(x)]^T$, onde z é o número de objetivos (COELLO; LAMONT; VELDHUIZEN, 2007).

Para os problemas que possuem apenas um objetivo, há somente um valor para a sua solução ótima (LUNA et al., 2013). No entanto, o conceito de solução ótima muda para os problemas multiobjetivos. Além de possuir várias soluções, a noção de otimalidade também varia nesse contexto. Boas soluções são aquelas que possuem um compromisso entre os objetivos, onde os valores dos mesmos estão balanceados (COELLO; LAMONT; VELDHUIZEN, 2007). Essas soluções fazem parte do Conjunto Ótimo de Pareto, indicando que um objetivo não pode ser melhorado sem que prejudique algum outro. Esse mesmo conjunto passa a ser chamado de Fronteira de Pareto quando plotado no espaço dos objetivos (GOLDBARG; GOLDBARG; LUNA, 2016). Matematicamente, define-se que (COELLO; LAMONT; VELDHUIZEN, 2007):

Definição 2.1.1 (Ótimo de Pareto) *Uma solução x^* é ótima de Pareto se não existir nenhum $x \in X$ tal que $f_z(x) < f_z(x^*)$ para todo $z \in K = \{1, 2, ..., k\}$. Essa definição mostra que se x^* é ótimo de Pareto então não existe nenhum vetor x que irá melhorar um objetivo sem causar, simultaneamente, a piora de um outro.*

Figura 1 – Fronteira de Pareto gerada a partir de uma instância do SPSP.



Fonte: Adaptado de (WIKIPÉDIA, 2018).

Definição 2.1.2 (Dominância de Pareto) *É dito que uma solução* $u = (u_1, u_2, ..., u_z)$*, domina uma outra solução* $v = (v_1, v_2, ..., v_z)$ *se e se somente se para todo* $z \in K = \{1, 2, ..., k\}$ *tem-se que* $f_z(u) \leq f_z(v)$ *e* $\ni z : f_z(u) < f_z(v)$.

Definição 2.1.3 (Fronteira de Pareto) *É a imagem de um determinado conjunto de soluções ótimo de Pareto no espaço dos valores objetivos.*

A Figura 1 ilustra uma fronteira de Pareto. Ela está representada por um plano cartesiano pois está relacionada um problema (de maximização) com dois objetivos. Os pontos em vermelho (A, B, ..., H) fazem parte da fronteira de Pareto. Esses pontos estão de acordo com a definição 2.1.2, uma vez que nenhum ponto dessa fronteira domina o outro por completo, isto é, há, ao menos, um objetivo no qual ele não é melhor do que outra solução pertencente ao mesmo conjunto. (ANDRADE; SILVA; BRITTO, 2018).

2.2 O Problema do Escalonamento em Projeto de Software

O projeto de um software é composto de tarefas e funcionários, que em diferentes cenários e ambientes de trabalho, se organizam e planejam diferentes formas de fazer um produto. Independentemente da demanda e do mercado alvo, é crucial que antes do projeto haja uma organização e divisão sobre quem irá fazer o que durante o desenvolvimento do software, a fim de respeitar todos os prazos e orçamentos (SOMMERVILLE, 2015). Essa divisão é comumente chamada de fase de escalonamento, onde cada funcionário é associado a uma ou mais tarefas e é definida sua participação na mesma. Tal escalonamento deve respeitar, também, a capacidade (habilidade ou contratual) de cada funcionário.

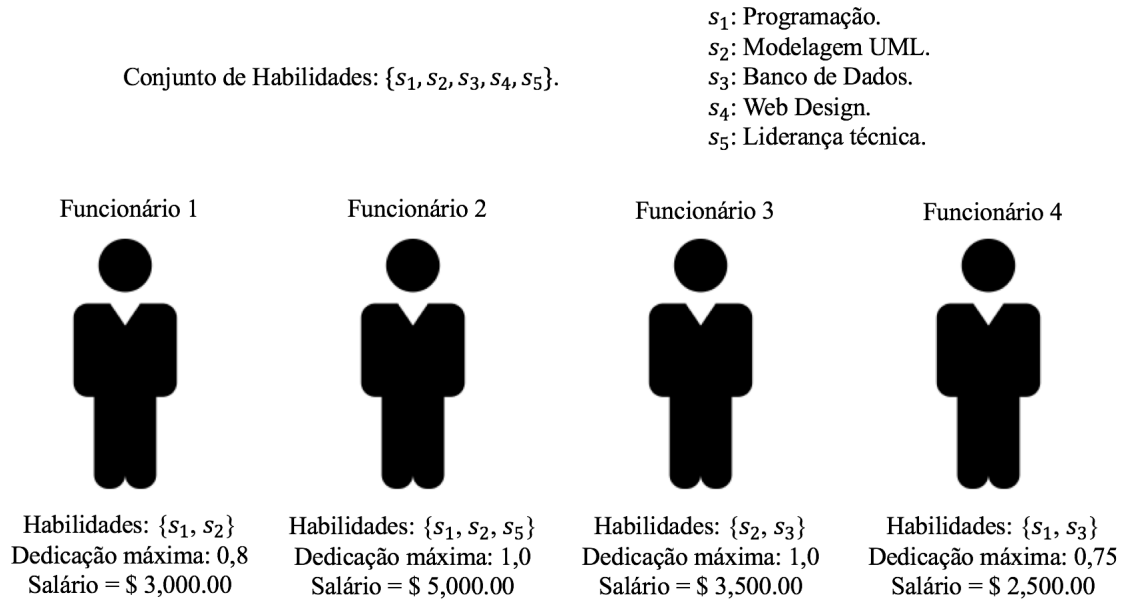
A tarefa de apresentar um escalonamento é realizada pelo líder do projeto. Segundo Pressman (2014), o líder deve fazer um escalonamento de tal forma que seja possível para ele controlar e monitorar o progresso do projeto. Sem um escalonamento formalmente definido, é impossível fazer isso caso o projeto seja complexo demais (PRESSMAN, 2014). Chen e Zhang (2013b) mostram que na China, por exemplo, mais de 40% dos projetos que falharam tinham fracos planos de projeto e funcionários. Já (GROUP, 2015) relatou que somente 29% de 50.000 projetos do ano de 2015, obtiveram sucesso. O relatório ainda mostra que, se avaliados por tamanho, apenas 2% dos projetos considerados grandes puderam chegar ao fim do desenvolvimento de forma satisfatória.

O problema de escalonar pessoas a tarefas em um projeto de software é estudado na literatura e é conhecido como o Problema do Escalonamento em Projeto de Software - *Software Project Scheduling Problem* (SPSP). O SPSP é uma subárea dentro da área relacionada a desenvolvimento de software, a tarefa de gestão de um projeto de software (FERRUCCI; HARMAN; SARRO, 2014), (PEIXOTO; MATEUS; RESENDE, 2014). Segundo Pressman (2014), a atividade de escalonar distribui o esforço estimado ao longo de toda a duração do desenvolvimento, alocando esse esforço em tarefas específicas do ciclo do desenvolvimento. Portanto, o SPSP consiste em associar um conjunto de funcionários a determinadas tarefas, seguindo regras e visando o maior lucro possível.

Como exemplo, considere as Figuras 2 e 3, onde ambas representam um possível cenário para um projeto de software. A Figura 2 elenca o conjunto de funcionários designados para o projeto. Cada funcionário possui habilidades particulares (o funcionário 1, por exemplo, possui habilidades de programação e modelagem UML), assim como uma dedicação ao trabalho e seu salário. Para as tarefas, na Figura 3, a representação é similar. Cada tarefa possui um custo associado a ela, e habilidades necessárias para a sua realização. Tomando como exemplo a tarefa 1, seu custo de produção é estimado em $\$15$ unidades monetárias, possuindo a programação e a modelagem UML como habilidades necessárias. Nesse caso, o funcionário 1 é capaz de realizá-la.

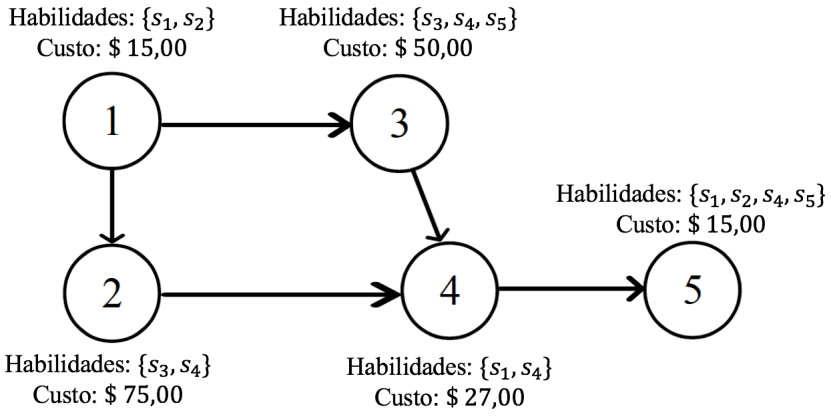
Uma dada solução para o SPSP está representada na forma de uma matriz, denominada de Matriz de Dedicação (DM). Nela, cada célula indica o quanto um funcionário irá dedicar-se para uma tarefa, em um dia de trabalho. Linhas representam o funcionário, enquanto que as colunas representam as tarefas. A Figura 4 ilustra uma matriz de dedicação. Nela, por exemplo, o Funcionário 3 dedica $100,0\%$ do seu dia de trabalho ao projeto, enquanto que o Funcionário 4 gasta $95,0\%$ de seu dia de trabalho no projeto. Se o valor de uma célula é $0,0$, então quer dizer que o funcionário em questão não trabalha nessa tarefa. Como o objetivo de todo líder de projeto é de que o tempo de desenvolvimento seja o menor possível, gastando-se uma quantidade mínima de recursos (ALBA; CHICANO, 2007), o escalonamento deve ser feito de forma ótima, mesmo que sujeito a várias restrições, que pode ser falta de habilidade por parte dos funcionários a até mesmo excesso de carga de trabalho. A Tabela 1 sumariza todas as variáveis de uma instância do SPSP. A partir delas, é possível computar a duração do projeto,

Figura 2 – Representação de um funcionário e suas propriedades.



Fonte: Próprio autor.

Figura 3 – Representação de uma tarefa e suas propriedades.



Fonte: Próprio autor.

bem como seu custo previsto.

Para a duração do projeto, primeiro deve-se calcular a duração individual de cada tarefa (Equação 2.1) e, em seguida, definir seu início e fim. Para as tarefas que não possuem nenhuma outra como pré-requisito, seu início se dá no início do projeto (instante 0), enquanto que as que necessitam da execução de outras tem início no instante em que a última predecessora acaba. O fim, portanto, é calculado como sendo a soma do instante do inicio da execução da tarefa, com a sua duração. Dessa forma, a duração total do projeto é dada como o instante em que a última

Figura 4 – Matriz de dedicação de um escalonamento.

	Tarefa 1	Tarefa 2	Tarefa 3	Tarefa 4	Tarefa 5
Funcionário 1	0,00	1,0	0,34	0,20	0,40
Funcionário 2	0,10	0,10	0,10	0,40	0,30
Funcionário 3	0,25	0,25	0,25	0,15	0,10
Funcionário 4	0,75	0,05	0,05	0,05	0,05

Fonte: Próprio autor.

Tabela 1 – Tabela de variáveis usadas no SPSP.

Variável	Significado
E	Quantidade total de funcionários
T	Quantidade total de tarefas
$G(V, E)$	Grafo de precedência de tarefas
e^{skills}	Conjunto de habilidades do funcionário
e^{sal}	Salário do funcionário
e^{over}	Excesso de trabalho de funcionário
t^{skills}	Conjunto de habilidades necessárias para a realização da tarefa
t^{cost}	Custo total para a realização da tarefa
t^{ini}	Instante inicial de uma tarefa
t^{end}	Instante final de uma tarefa
t^{dur}	Tempo de duração de uma tarefa
p_{cost}	Custo total do projeto
p_{over}	Excesso de trabalho total do projeto
$X = (m_{ij})_{E \times T}$	Matriz de dedicação dos funcionários ao projeto

Fonte: Próprio autor.

tarefa acaba, isto é, o maior tempo final calculado.

$$t_j^{dur} = \frac{t_j^{cost}}{\sum_{i=1}^{E} x_{ij}} \tag{2.1}$$

O custo do projeto é a soma de todas as taxas pagas aos funcionários, de acordo com suas dedicações ao projeto. As taxas são calculadas multiplicando o salário dos funcionários pelo tempo gasto no projeto. O tempo gasto no projeto é a soma de suas dedicações multiplicada pela duração de cada tarefa. Formalmente:

$$p_{cost} = \sum_{i=1}^{E} \sum_{j=1}^{T} e_i^{sal} \cdot x_{ij} \cdot t_j^{dur} \tag{2.2}$$

Por fim, para que uma solução seja dada como válida, devem ser satisfeitas três restrições. A primeira é de que toda tarefa deve ser realizada por algum funcionário, de tal forma

que todas sejam completadas. Isto é:

$$\sum_{i=1}^{E} x_{ij} > 0 \; \forall j \in \{1, 2, ..., T\} \tag{2.3}$$

Segundo, a união dos conjuntos de habilidades de todos os funcionários associado a uma tarefa deve conter ou ser o conjunto de habilidades necessárias para a realização de uma tarefa:

$$t_j^{skills} \subseteq \bigcup_{\{i|x_{ij}>0\}} e_i^{skills} \; \forall j \in \{1, 2, ..., T\} \tag{2.4}$$

Como nenhum funcionário deve trabalhar acima de seu limite, o excesso de trabalho do projeto (*overwork*) deve ser 0 ($p_{over} = 0$). Para verificar essa restrição, primeiro calcula-se a função de trabalho (τ) de cada funcionário, em relação ao tempo de execução do projeto:

$$e_i^{work}(\tau) = \sum_{\{j|t_j^{start} \leq \tau \leq t_j^{end}\}} x_{ij} \tag{2.5}$$

O funcionário excede o seu limite de trabalho caso $e_i^{work}(\tau) > e_i^{maxded}$ em qualquer instante τ da execução do projeto. O excedente total do funcionário (e_i^{over}) é representado por:

$$e_i^{over} = \int_{\tau=0}^{\tau=p_{dur}} ramp(e_i^{work}(\tau) - e_i^{maxded}) d\tau \tag{2.6}$$

Onde $ramp$ é uma função definida por:

$$ramp(x) = \begin{cases} x, \; se \; x > 0 \\ 0, \; se \; x \leq 0 \end{cases} \tag{2.7}$$

2.3 Engenharia de Software Baseada em Busca

A Engenharia de Software Baseada em Busca - *Search-Based Software Engineering* foi introduzida em Harman e Jones (2001). A SBSE investiga técnicas de buscas, como meta-heurísticas, para a solução de problemas da engenharia de software, muitos deles resolvidos de forma *ad hoc*. Desde sua concepção, a SBSE vem crescendo e atraindo mais pesquisas sobre novas modelagens e novas técnicas de busca de soluções (HARMAN, 2012). De acordo com (HARMAN; JONES, 2001), apenas dois quesitos são necessários para estudar um problema da engenharia de software na área de SBSE:

1. Uma representação formal do problema.

2. A definição do quão boa é uma solução através dos valores do(s) objetivo(s) (*fitness*) .

O item 1 refere-se a como o problema pode ser representado de tal forma que possa ser aplicado técnicas computacionais, tais como algum cálculo sobre a representação. São exemplos de representações um vetor, ou uma matriz. Já a função objetivo, item 2, é responsável por guiar o processo de busca, uma vez que é ela quem quantifica, através de métricas (HARMAN; CLARK, 2004), o quão boa é a solução em relação a um referencial, que pode ser simplesmente o conjunto disponível de soluções.

A modelagem proposta por Alba e Chicano (2007), objeto da seção 2.2, para o Problema do Escalonamento em Projeto de Software se encaixa nos critérios 1 e 2 da SBSE. A representação formal permite gerar a matriz de dedicação, como uma solução, e a partir da mesma, os *fitness* são representados pelos quatro objetivos: duração, custo, estabilidade e robustez. A busca por soluções se dá pela aplicação de meta-heurísticas, descritas a seguir.

2.4 Meta-heurísticas

O Problema do Escalonamento em Projeto de Software é um problema de otimização, onde, dentre todos os objetivos, deseja-se otimizá-los de tal forma que, escolhidas suas variáveis de decisão, sejam obtida soluções que apresentem um bom compromisso entre eles. No entanto, problemas de otimização impõem um grande desafio para a computação, sobretudo no que se trata a complexidade para oferecer soluções para tais problemas.

Desde que muitos problemas de otimização, como o SPSP (CRAWFORD et al., 2014), são classificados na literatura como pertencentes a classe $\mathcal{NP}$-Difícil, não há evidências de algoritmos exatos que possam resolvê-los sem consumir um tempo (ou memória) que seja exponencial em relação ao tamanho da entrada do mesmo, a menos que $\mathcal{P} = \mathcal{NP}$ (GAREY; JOHNSON, 1979). Diante do esforço em resolver problemas dessa complexidade com um tempo de execução viável, as heurísticas são aplicadas. Assim sendo, define-se heurísticas da seguinte forma:

Definição 2.4.1 (Heurísticas) *Uma heurística é uma técnica de solução de problema onde a solução parcial mais apropriada no estado atual é selecionada a partir de métodos que empregam regras comparativas. (COELLO; LAMONT; VELDHUIZEN, 2007)*

Segundo (GOLDBARG; GOLDBARG; LUNA, 2016), as heurísticas visam alcançar soluções avaliadas como aceitáveis para um dado problema utilizando um esforço computacional consideravelmente aceitável, garantindo, em determinadas condições, a otimalidade da solução gerada. Isso acontece, pois, os métodos heurísticos sistematicamente selecionam soluções aparentemente boas, segundo seus métodos de seleções empregados. Soluções boas para tais métodos são aquelas que, localmente, se destacaram em relação as outras, e valem a pena

serem exploradas. Esses métodos são oriundos de várias inspirações, podendo ser uma simples comparação entre as soluções candidatas ou até mesmo a computação de alguma métrica estatística qualquer. Mas, devido a enorme quantidade de problemas e variantes destes na natureza, um conjunto de heurísticas pode ser bom para uma classe de problema mas péssimo para outros (LAM; LI, 2010).

A busca por algoritmos que possuem um maior poder de generalização resultaram em métodos abstratos que estão em um nível maior do que o problema ao qual será aplicado. Esses algoritmos receberam o nome de meta-heurísticas e representam o atual estado da arte de otimização (GOLDBARG; GOLDBARG; LUNA, 2016). O prefixo *meta* sugere uma abstração do algoritmo na solução do problema. De fato, segundo (GOLDBARG; GOLDBARG; LUNA, 2016), uma meta-heurística pode ser vista como método que guia o processo de computação de uma ou mais heurísticas para a solução de um problema (apud DORIGO; STÜZLE, 2004). Meta-heurísticas são, então, definidas da seguinte forma:

Definição 2.4.2 (Meta-heurísticas) *Meta-heurísticas são estratégias de alto nível que guiam outros processos de baixo nível, como heurísticas, buscando soluções viáveis em contextos com domínios complexos. (COELLO; LAMONT; VELDHUIZEN, 2007)*

2.4.1 Meta-heurísticas Evolucionárias

A meta-heurística usada neste trabalho é considerada pertencente à classe de meta-heurísticas evolucionárias - *Multiobjective Evolutionary Algorithms* (MOEA). Essa categoria de algoritmos possui inspirações oriundas de fenômenos da natureza, tais quais biológico, químico, etc (GOLDBARG; GOLDBARG; LUNA, 2016). É característico dessa classe basear seu conjunto solução como uma população, onde cada indivíduo dessa população é uma solução por si própria. O método de busca das meta-heurísticas evolucionárias pode ser facilmente guiado pelo domínio do problema sem que precise ser modificado (COELLO; LAMONT; VELDHUIZEN, 2007). Esse aspecto transformou-se em uma vantagem para as MOEA, visto que por conta disso seu desenvolvimento se tornou simples, bem como seu entendimento e sua visualização. Harman (2012) cita que MOEA é o tipo de meta-heurística mais usado na SBSE. Portanto, todas as MOEA compartilham um esquema algorítmico, mostrado no Algoritmo 1,

com poucas variações.

Algoritmo 1: Pseudocódigo de uma Meta-heurística Evolucionária

```
   Saída: Fronteira de Pareto.
1  início
2      população ← inicialização da população, conjunto inicial vazio;
3      Criação de invíduos que irão compor as primeiras soluções de população;
4      Avaliação de cada indivíduo ∈ população;
5      enquanto critério de parada não atingido faça
6          Atualização da população atual utilizando operadores evolucionários;
7          população_descendente ← geração de invidíduos a partir da população atual;
8          Avaliação de cada indivíduo ∈ população_descendente;
9          população ← seleção dos melhores indivíduos de população_descendente;
10     fim
11     retorna Melhores indivíduos gerados
12 fim
```

No algoritmo 1, uma MOEA começa o processo de busca a partir da criação de uma população inicial (linha 3), que pode ser gerada de diversas formas, podendo ser aleatória ou com uma estratégia pré-definida. O próximo passo, na linha 4, é a atribuição à cada indivíduo da população um *fitness* correspondente, computado a partir das suas variáveis de decisão e de uma métrica qualquer. Assim, a meta-heurística inicia o processo de busca que se repete até que um critério de parada seja atingido (linha 5). O procedimento é repetido de forma similar aos passos iniciais, sendo a diferença consistindo em duas características: 1) operadores evolucionários (mutação, cruzamento, etc) são utilizados para modficar os indivíduos atuais e 2) os novos indivíduos, pertencentes a uma população descendente, são gerados a partir da população anterior à ela. Dessa forma, há um aprimoramento das soluções existentes e a MOEA mantém a capacidade de explorar novos locais de busca.

O passo descrito na linha 7 é o mesmo desempenhado na linha 3. Após a atribuição do *fitness*, os melhores indivíduos descendentes são selecionados e passam a compor a população inicial (linha 8). Ao final da execução, depois de atingir o critério de parada, é retornado um conjunto com os melhores indivíduos gerados durante o processo. Nele, nenhum indíviduo domina um outro, caracterizando, portanto, uma fronteira de Pareto.

2.4.2 Otimização Por Enxame de Partículas

A Otimização por Enxame de Partículas - *Particle Swarm Optimization* (PSO) (KENNEDY; EBERHART, 1995) - é uma meta-heurística baseada no comportamento social dos pássaros, quando estes buscam por comida. Devido ao seus bons resultados em problemas de vários tipos, sobretudo nos quais as variáveis são contínuas (REYES-SIERRA; COELLO, 2006), PSO é largamente usada na literatura, com diversas variantes, algumas delas adaptadas para problemas com muitos objetivos.

Um indivíduo de sua população é denominado de partícula, e o seu conjunto, enxame. Cada uma dessas partículas possui variáveis que computam a sua posição no espaço de busca e a velocidade da mesma. A posição de cada partícula é atualizada de acordo com a sua experiência e a sua vizinhança. Para calcular-se a distância, leva-se em conta a velocidade dessa partícula.

A qualidade de uma vizinhança tem um papel importante na qualidade de uma partícula. Além das variáveis já descritas, um indivíduo também possui variáveis relativas à posições de outras partículas (e dela mesma): $pbest$, $lbest$, $gbest$. A primeira armazena a melhor posição encontrada pela partícula. A segunda, a melhor posição encontrada em uma dada vizinhança. A terceira, a melhor posição encontrada dentre todas as partículas. Essas variáveis servem de apoio pra a atualização do(s) líder(es) de uma partícula. Um líder é um indivíduo usado para guiar outras partículas em direção a melhores regiões do espaço de busca.

Desde a primeira tentativa de adaptação do PSO para problemas com muitos objetivos, mais de trinta variantes, que possuem uma abordagem multi-objetiva, foram desenvolvidas. Dentre elas, destaca-se a *Speed-constrained Multi-objective PSO* (SMPSO) (NEBRO et al., 2009). Essa variante se diferencia das outras por incorporar um mecanismo de controle de velocidade à partículas, uma vez que esta poderia se tornar alta demais, resultando em movimentos errôneos das mesmas. Esse evento, de descontrole de velocidade, é chamado de explosão de enxame.

Além do mecanismo de controle de velocidade, a SMPSO implementa um operador de mutação, denominado de "fator turbulência", permitindo uma maior exploração do espaço de busca. Por último, para guardar as soluções não dominadas até um determinado instante, um arquivo (representado por uma lista) é implementado. O Algoritmo 2 mostra o ciclo da execução do SMPSO. Nota-se a semelhança com o Algoritmo 1, que esquematiza o funcionamento das meta-heurísticas evolucionárias.

Algoritmo 2: Pseudocódigo de uma Meta-heurística Evolucionária

```
Saída: Fronteira de Pareto.
início
    população ← inicialização da população, conjunto inicial vazio;
    líderes ← inicialização dos líderes, conjunto inicial vazio;
    Avaliação de cada indivíduo ∈ população;
    enquanto critério de parada não atingido faça
        Atualização da população atual utilizando operadores evolucionários;
        população_descendente ← geração de invidíduos a partir da população atual;
        Avaliação de cada indivíduo ∈ população_descendente;
        população ← seleção dos melhores indivíduos de população_descendente;
    fim
    retorna Melhores indivíduos gerados
fim
```

Todas essas técnicas implementadas fizeram a SMPSO produzir resultados competiti-

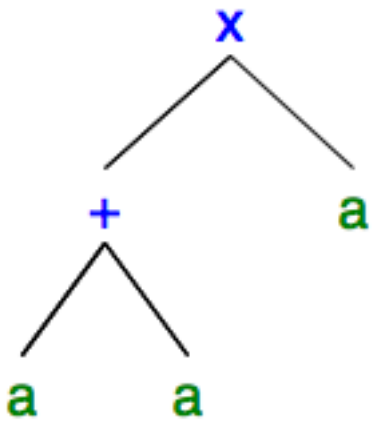
vos, quando comparados a outros algoritmos evolucionários clássicos da literatura. Portanto, constata-se a adequação e eficiência (em relação a outras variantes do PSO), para a realização dos experimentos desse projeto de pesquisa.

2.5 Programação Genética

A Programação genética (KOZA, 1992) é uma técnica usada para a geração automática de programas, os quais podem serem usados para solução de problemas. Sua representação mais comum é em forma de árvore, devido a sua natureza recursiva. Desde que GP é uma variante de Algoritmos Evolucionários, são herdados os operadores capazes de modificar ou gerar soluções. A Figura 5 ilustra a expressão $(a + a) \times a$ estruturada em forma de árvore. As variáveis (a) são chamados de folhas, uma vez que não possuem ramificações a partir deles, enquanto que os operadores $(+$ e $\times)$ são chamados de funções.

O procedimento de geração de uma nova solução é similar ao de uma algoritmo evolucionário. Primeiro, é gerada uma população aleatória. Em seguida, já no processo evolutivo, duas soluções são selecionadas para a aplicação dos processos de reprodução e mutação. Os métodos mais comuns para esses processos selecionam um ponto (nó na árvore) para a aplicação do operador, que servirá de referencial na mudança da estrutura da árvore. É comum que a estrutura da árvore seja alterada apenas na subárvore cuja raiz foi o ponto selecionado. Assim que a solução é gerada, seu *fitness* é computado e ela passa a competir com as outras soluções pela sobrevivência. Ao final, apenas a melhor solução é retornada.

Figura 5 – Árvore para a expressão $(a + a) \times a$



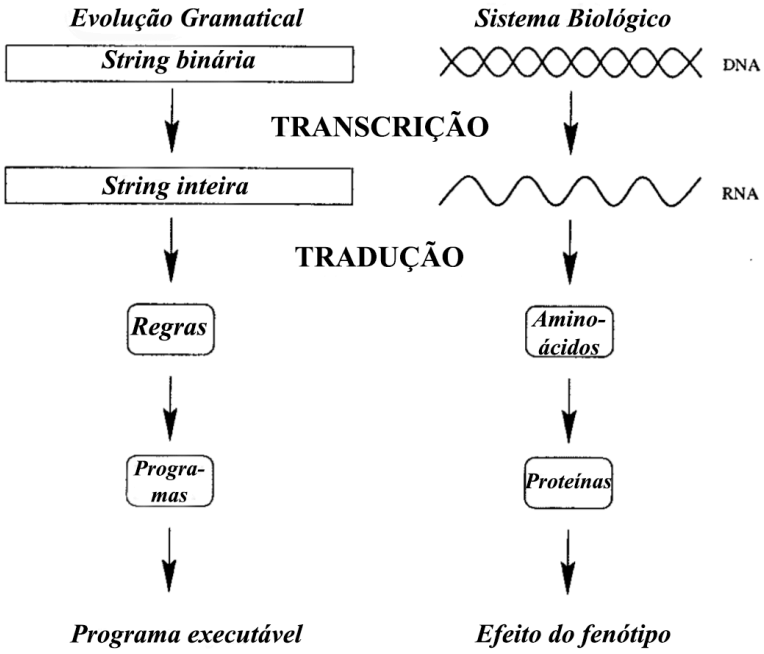
Fonte: Próprio Autor

2.6 Evolução Gramatical

A Evolução Gramatical, proposta por (O'NEILL; RYAN, 2001), é um tipo derivado da Programação Genética e, portanto, similar em sua execução. Contudo, enquanto que na Programação Genética o passo evolucionário se dá na árvore do programa (e, portanto, nos programas em si), na GE as alterações são feitas em *strings* (cromossomos), as quais são mapeadas em *arrays*, para a geração de programas. O processo de geração de um programa é denominado de *mapeamento genótipo-fenótipo*. Tal mapeamento se dá a partir da manipulação de *strings* binárias, as quais representam produções de uma gramática qualquer, escrita na forma de *Backus-Naur*.

A Figura 6 mostra como se dá o mapeamento, desde a *string* até o programa gerado. A *string* é análoga ao DNA biológico, e ambos passam pelo processo de transcrição, quando, no caso da GE, é gerado o *array* de *strings*. Para chegar ao *fenótipo*, o processo de tradução trata de mapear o valor de cada posição do *array* para as regras de produção da gramática, até que seja alcançado um terminal.

Figura 6 – Comparação entre os sistemas genéticos e a evolução gramatical.



Fonte: Adaptado de (O'NEILL; RYAN, 2001, p. 351).

Uma gramática na forma *Backus-Naur* (BNF) (NAUR et al., 1997) é expressa através de regras de substituições. Formalmente, uma gramática na BNF é descrita por uma tupla $(V, \sum, R, S)$, onde V é um conjunto finito de *variáveis*, $\sum$ é um conjunto finito de *terminais*, R é um conjunto finito de *produções*, as quais podem ser variáveis e terminais, e S sendo a *variável inicial* (SIPSER, 2012). Considerando a gramática 2.1, têm-se que $V = \{< expressao >, < termo >, < fator >\}$, $\sum = \{a, +, \times, (,)\}$, e $S = \{< expressao >\}$. Através dela, é

possível formar a *string* que é a representação da expressão usada na Figura 5. As substituições realizadas estão apresentadas na árvore de transformação da Figura 7.

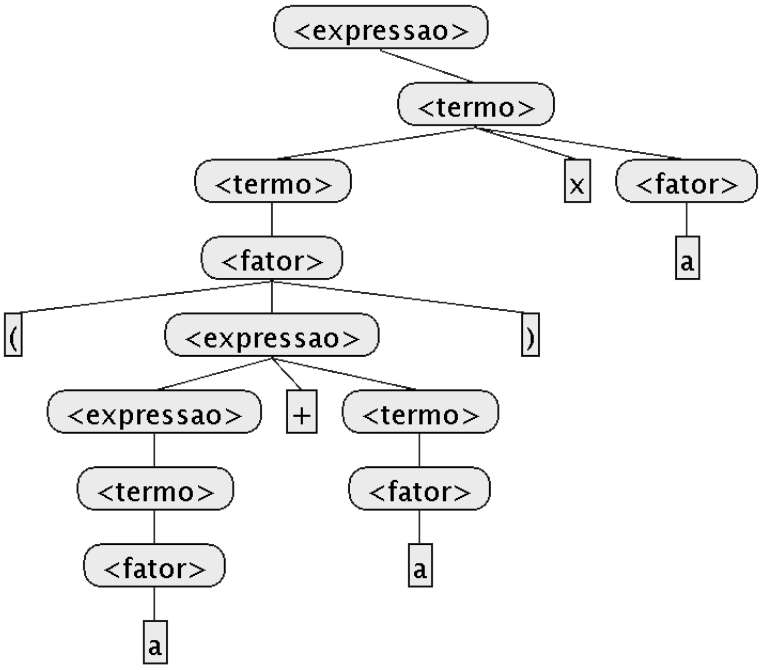
Gramática 2.1 – Gramática geradora da expressão $(a + a) \times a$.

⟨expressao⟩ ::= ⟨expressao⟩ '+' ⟨termo⟩ | ⟨termo⟩

⟨termo⟩ ::= ⟨termo⟩ 'x' ⟨fator⟩ | ⟨fator⟩

⟨fator⟩ ::= (⟨expressao⟩) | a

Figura 7 – Árvore de transformações para a geração da expressão 5



Fonte: Próprio Autor.

Uma solução na GE é comumente representado por um array de *bits* ou inteiros. Dessa forma, quaisquer operadores evolucionários, que computam sobre essas representações, podem ser aplicados nessa fase do processo evolutivo. É comum que pesquisadores que trabalham com GE ignorem a fase de transcrição da solução (Figura 6) e simplesmente trabalhem diretamente com o *array* de inteiros (MARIANI, 2017). Uma representação de solução está expressa pelo *array* da Figura 8. Com essa mesma solução, é possível gerar a expressão $(a + a) \times a$ através de seu mapeamento genótipo-fenótipo, usando a gramática 2.1.

Figura 8 – Uma solução expressa por um array de inteiros.

333	700	3	2	10	89	1001	13	55	43	3

Fonte: Próprio autor.

Considerando a indexação das posições a partir do 0, o seu respectivo valor, 333, e a produção inicial $< expressao >$, o primeiro passo a ser tomado é fazer a contagem de quantas alternativas a produção em questão possui, duas no total. Em seguida, é computado o módulo do valor da posição em questão, do *array*, com a quantidade computada de opções. Assim, têm-se que $333\%2 = 1$, e a opção $< termo >$ é escolhida. Dessa forma, a transformação parcial é:

Figura 9 – Expressão após a primeira transformação

```
<expressao> -> <termo>
```

Fonte: Próprio autor

O processo se repete até que todas as posições do *array* tenham sido usadas, ou até que a transformação tenha chegado até o fim. Portanto, a segunda posição, de valor 700, irá determinar a transformação que ocorrerá na variável mais a esquerda $< termo >$. Como $< termo >$ possui duas opções, $700\%2 = 0$, resultando em:

Figura 10 – Expressão após segunda transformação

```
<termo> -> <termo> x <fator>
```

Fonte: Próprio autor.

Após percorrer todas as posições, e aplicando as transformações sempre a variável mais esquerda possível da expressão, o resultado final será a expressão $(a + a) \times a$. O *array* da Figura 8 produziu a transformação usando exatamente todos os seus valores. No entanto, caso todas as posições não tivessem sido usadas, elas simplesmente seriam ignoradas. Se por ventura todas as posições tivessem sido usadas, mas a expressão estivesse completa, os valores seriam reusados e o processo continuaria até a expressão ter sido validada totalmente.

2.7 Hiper-Heurísticas

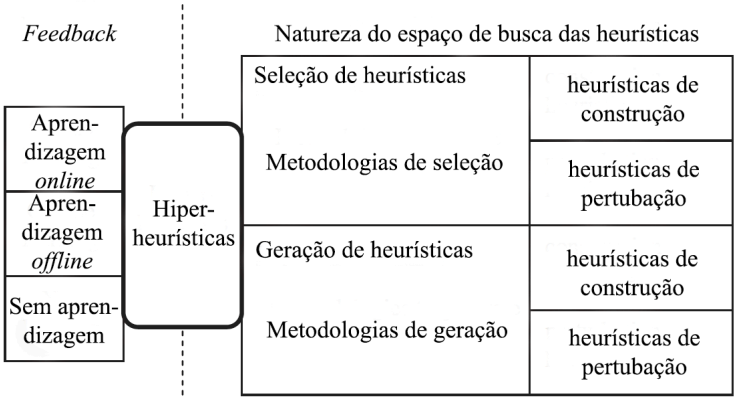
Hiper-heurísticas são algoritmos que são capazes de gerar outras heurísticas. Podem ser geradas simples heurísticas, meta-heurísticas, ou até mesmo novas hiper-heurísticas (MARIANI, 2017). Ao invés de procurar boas soluções para um dado problema, as hiper-heurísticas focam sua busca em gerar novas heurísticas que melhor se ajustam para o problema, alcançando melhores soluções à medida que a busca ocorre. Uma vez que o motor de busca de uma meta-heurística é altamente dependente de seus parâmetros de entrada, o foco da busca das hiper-heurísticas é na definição de parâmetros que maximizem a qualidade das soluções encontradas pelos algoritmos gerados (BURKE et al., 2013).

Dessa forma, observa-se que Hiper-heurísticas devem ser genéricas o suficiente, ao passo de que o domínio do problema não pode afetar o seu desenvolvimento. Assim, surgem diferentes maneiras de classificar uma hiper-heurística, as quais dependem da sua forma de busca e do retorno gerado (onde o mesmo é usado no processo de busca) (BURKE et al., 2013). Com relação a forma em que a busca se dá, hiper-heurísticas podem ser divididas entre as que *geram* heurísticas e as que *selecionam* heurísticas. Enquanto que o primeiro tipo aplica técnicas de geração, o segundo usa informações já existentes para selecionar heurísticas.

Essa forma de classificação pode ainda ser estendida ao se referir a metodologia usada para a obtenção de soluções, as quais são classificadas como construtivas ou perturbativas. Heurísticas construtivas funcionam de forma similar a MOEA, inicialmente com um conjunto vazio de soluções, e a cada iteração novas soluções surgem ou são aprimoradas. As perturbativas começam com soluções geradas inicialmente e vão alterando-as ao longo da execução do algoritmo.

Por fim, considerando o retorno dado pela hiper-heurística, a classificação varia entre *online* e *offline*. Uma hiper-heurística *online* usa os resultados já gerados para guiar-se durante a busca, de forma similar a outras MOEAs, como os algoritmos genéticos, busca tabu, e a meta-heurística baseada em enxame de partículas. Por outro lado, quando a busca se dá de forma *offline*, é comum que o aprendizado se dê apenas durante a fase de treinamento, antes de resolver os problemas reais. Um exemplo de busca *offline* é a Programação Genética (SABAR et al., 2013). A Figura 11 esquematiza as categorias existentes nas hiper-heurísticas.

Figura 11 – Categorização de Hiper-heurísticas.



Fonte: Adaptado de (BURKE et al., 2013, p. 3).

2.7.1 Esquematização de uma hiper-heurística

Como já especificado, hiper-heurísticas podem gerar meta-heurísticas de forma não supervisionada. Uma maneira é a partir da busca por parâmetros que otimizem os resultados obtidos pela devida configuração. Dessa forma, hiper-heurísticas também compartilham o es-

quema apresentado no algoritmo 1. A cada atualização da população, a função objetivo da população atual é computada, sobrevivendo apenas as melhores soluções. Estas, por sua vez, representam uma parametrizagem em cada uma das meta-heurísticas geradas a partir de cada uma delas.

A comparação entre as meta-heurísticas geradas pode valer-se dos princípios da SBSE. Mais especificamente, o item 2 (página 17) indica que as funções objetivos podem ser comparadas a partir de métricas. Assim, assume-se a fronteira de Pareto FP resultante da meta-heurística gerada pela parametrizagem (indivíduo) da hiper-heurística. FP é, então, avaliada por uma métrica I que retornará um valor $I(FP)$. Caso I seja uma métrica de maximização, as meta-heurísticas que gerarem os maiores valores $I(FP)$ prevalecerão. O caso de minimização é análogo. O algoritmo 3 sumariza toda a esquematização detalhada.

Algoritmo 3: Esquema de funcionamento de uma Hiper-heurística.

```
Entrada: Arquivo contendo a gramática (GF)
Saída: Melhores parametrizagens geradas pela gramática
1  início
2      população_hiper ← inicialização da população da hiper-heurística;
3      meta-heurísticas ← mapeamento de população_hiper utilizando a GF;
4      população_meta ← execução das meta-heurísticas;
5      Para cada FP ∈ população é assinalado o valor retornado por I(FP);
6      enquanto critério de parada não atingido faça
7          parents ← seleção dos pais;
8          descendentes ← cruzamento dos pais;
9          Aplicação do operador de corte nos descendentes;
10         Aplicação do operador de duplicação nos descendentes;
11         Aplicação do operador de mutação nos descendentes;
12         meta-heurísticas-descendentes ← mapeamento dos descendentes utilizando
             a GF;
13         Para cada FP ∈ meta-heurísticas-descendentes é assinalado o valor
             retornado por I(FP);
14         população_meta ← seleção dos melhores indivíduos de descendentes;
15     fim
16     retorna Melhores indivíduos gerados
17 fim
```

3 Metodologia

O projeto de pesquisa foi dividido em fases as quais eram compostas por diferentes atividades. Com exceção do item Fase 3.1., as atividades são descritas a seguir na ordem em que foram realizadas.

3.1 Fases do Projeto

Fase 1. **Revisão Bibliográfica da literatura, segundo Kitchenham (2004) e Kitchenham e Charters (2007).**

- Fase 1.1. Coleta da literatura relacionada a SBSE;
- Fase 1.2. Coleta da literatura relacionada ao SPSP e suas variantes;
- Fase 1.3. Coleta da literatura sobre meta-heurísticas, em especial, da Otimização por Enxame de Partículas, Programação Genética e Evolução Gramatical.

Fase 2. **Implementação e adaptação dos algoritmos escolhidos e estudados.**

- Fase 2.1. Implementação do SPSP, segundo Alba e Chicano (2007);
- Fase 2.2. Escrita da gramática do algoritmo de Evolução Gramatical;
- Fase 2.3. Implementação do código para Evolução Gramatical;
- Fase 2.4. Adaptação da meta-heurística SMPSO para o *framework* de evolução gramátical.

Fase 3. **Realização do experimentos e conclusões.**

- Fase 3.1. Coleta de base de dados públicas para a realização dos experimentos;
- Fase 3.2. Realização dos experimentos.

Fase 4. **Escrita do relatório final e de artigos científicos.**

3.2 Cronograma de Atividades e Alterações no Projeto

Durante a realização do projeto, segundo a metodologia apresentada, um cronograma de atividades foi proposto, seguido e apresentado na Tabela 2. Ao longo do projeto, alterações no projeto foram feitas, tendo em vista que o autor já havia desenvolvido um ano de Iniciação Científica no tema, explorando a meta-heurística NCRO. Dessa forma, optou-se por explorar não somente a meta-heurística de Enxame de Partículas para o problema, mas estender este trabalho com a investigação de possíveis melhorias e adaptações quando esta meta-heurística é aplicada a hiper-heurísticas.

Tabela 2 – Cronograma do projeto.

Fase	Atividade	Ago	Set	Out	Nov	Dez	Jan	Fev	Mar	Abr	Mai	Jun	Jul
1	1	X											
	2	X											
	3	X											
2	1		X	X									
	2				X	X	X						
	3				X	X	X	X					
	4								X	X	X	X	
3	1	X											
	2											X	X
4	1											X	X

4 Experimentos

O presente capítulo detalha os passos escolhidos para a realização dos testes, procurando validar a hipótese proposta em 1.1. Para tanto, serão descritas as instâncias utilizadas nos experimentos. Elas representam um projeto de software. Para as hiper-heurísticas, será explicada a parametrização utilizada, assim como os seus operadores e a gramática usada para geração do SMPSO. Finalmente, os resultados são comparados a partir da métrica Hipervolume, a qual avalia a qualidade da fronteira de Pareto gerada.

4.1 Instância do SPSP

Uma instância descreve um cenário em um projeto de software. A instância escolhida foi utilizada pela primeira vez em Alba e Chicano (2007), e representa um projeto de software composto por 16 tarefas e 8 funcionários. Cada tarefa e funcionário possui características específicas, assim como detalhado em 2.2. A escolha deu-se a partir de um balanço entre tempo de execução e custo computacional, visto que hiper-heurísticas demandam um grande processamento.

4.2 Parametrização da Hiper-heurística

Uma vez que a hiper-heurística foi construída baseada em um algoritmo genético, seus operadores foram determinados de acordo com o esquema detalhado na Seção **[S]**. Eles estão definidos na tabela 3.

Tabela 3 – Parâmetros de execução da hiper-heurística.

Parâmetro	Tipo
Seleção	Binary Tournament
Recombinação	Single Point Crossover Variable Length
Mutação	Integer Mutation Prune Mutation Duplication Mutation
Tamanho da população	70 indivíduos
Número de avaliações de objetivo	10.000 avaliações

O operador *Binary Tournament* é responsável pela seleção de genes (soluções). Ao selecionar duas soluções de forma aleatória, são comparados entre elas atributos como número de restrições violadas, valor da função objetivo, entre outros. O cruzamento de soluções dá-se através do operador *Single Point Crossover Variable Length* através do sorteio aleatório de uma posição onde as variáveis anteriores a ela recebem o valor de um indivíduo e todas as

posteriores do segundo indivíduo. *Prune Mutation* e *Duplication Mutation* desempenham o papel de diminuir (cortar) e aumentar (replicar) a quantidade de variáveis em uma solução, respectivamente.

4.3 Gramática utilizada

A gramática BNF utilizada nos experimentos é descrita na gramática 4.1. A produção inicial, $< SMPSO >$, determina quais serão os parâmetros que serão gerados durante o processo de busca. Em ordem: tamanho da população ($< swarmSize >$), operadores de mutação (tipo de operador $< mutationOperatorPSO >$ e probabilidade $< mutationProbability >$), tipo e tamanho do arquivo ($< archive >$ e $< archiveSize >$) e metodologia de inicialização da população ($< initialization >$).

Todos os valores das produções foram escolhidos para compor a gramática baseado em testes realizados pela literatura, onde estes são referenciados como valores padrões. O atributo N denota a quantidade de variáveis de decisões do problema.

Gramática 4.1 – Gramática utilizada nos experimentos para a geração de meta-heurísticas.

⟨SMPSO⟩ ::= ⟨swarmSize⟩ ⟨matingOperatorsPSO⟩ ⟨archive⟩ ⟨initialization⟩

⟨swarmSize⟩ ::= 50 | 75 | 100 | 125

⟨matingOperatorsPSO⟩ ::= ⟨mutationOperator⟩ ⟨mutationProbability⟩

⟨mutationOperator⟩ ::= Polynomial Mutation
| Uniform Mutation
| Simple Random Mutation

⟨mutationProbability⟩ ::= 0.01 | 0.02 | 0.05 | 0.1 | 0.2 | 0.5 | 0.7 | 0.8 | 0.9 | 1.0 | 1.0 / N

⟨archive⟩ ::= ⟨fitnessAssignment⟩ ⟨archiveSize⟩

⟨archiveSize⟩ ::= 0 | N | N * 1.5 | N * 2

⟨fitnessAssignment⟩ ::= ⟨rankingStrategy⟩ ⟨diversityStrategy⟩

⟨rankingStrategy⟩ ::= | Dominance Rank | Dominance Strength | Dominance Depth
| Raw Fitness

⟨diversityStrategy⟩ ::= | Crowding Distance | K-th Nearest Neighbor | Adaptive Grid | Hypervolume Contribution

⟨initialization⟩ ::= Random | Parallel Diversified Initialization

O número máximo de iterações do SMPSO gerado irá variar toda vez que o parâmetro $< swarmSize >$ também variar. Isso acontece, pois, ao fixar um número máximo de iterações, a comparação tornar-se-ia injusta entre os indivídos da hiper-heurística. Um indíviduo que gerasse um SMPSO com um maior número de partículas e um número maior de iterações teria uma possibilidade maior de maximizar a sua função objetivo de acordo com alguma métrica, como o hipervolume (seção 4.5.1), por exemplo. Portanto, a quantidade de iterações de um SMPSO gerado é computado a partir da equação 4.1. Nela, a quantidade máxima de iterações é inversamente proporcional ao valor de $< swarmSize >$.

$$iteracoes = \frac{25000}{< swarmSize >} \tag{4.1}$$

4.4 Estratégia adotada

A adoção de uma estratégia possibilita a comparação dos algoritmos e a averiguação da questão de pesquisa proposta na seção 1.2: **É possível obter melhores resultados para o SPSP utilizando hiper-heurísticas?** A estratégia teve em mente que os resultados obtidos são oriundos de diferentes fontes. Em primeiro lugar, a partir de 9 execuções independentes da hiper-heurística, obteve-se 9 SMPSOs com configurações diferentes.

Por outro lado, para comparar com uma situação real, foram utilizadas soluções já obtidas por outras meta-heurísticas, estas geradas em trabalhos anteriores do autor deste projeto (ANDRADE; SILVA; CARVALHO, 2017). São elas: a *Nondominated Sorting Genetic Algorithm II* (NSGA-II) (DEB et al., 2002) e a *Nondominated Sorting Chemical Reaction* (NCRO) (BECHIKH; CHAABANI; SAID, 2015). As suas respectivas parametrizações estão descritas na tabela 4, sendo que o símbolo $-$ significa que tal algoritmo não possui tal parâmetro em sua configuração.

4.5 Comparação dos resultados

Os resultados foram comparados utilizando a métrica hipervolume (BEUME et al., 2009). Ela serve para computar a qualidade das fronteiras geradas por todas as meta-heurísticas utilizadas neste trabalho e está detalhada na seção 4.5.1. Já o teste de Kruskal-Wallis foi usado para determinar se os hipervolumes podem ser considerados diferentes estaticamente. As duas métricas estão detalhadas a seguir.

Tabela 4 – Parametrização das meta-heurísticas previamente utilizadas.

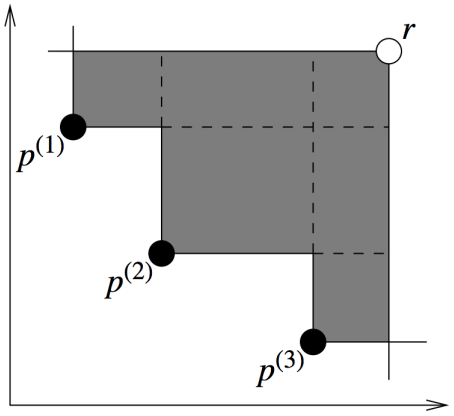
Parâmetro	NSGA-II	NCRO
População	100 indivíduos	100 indivíduos
Mutação	Polinomial, $p = 1,0/L$	Polynomial, $p = 1,0/L$
Recombinação	Cruzamento binário, $p = 0,9$	Cruzamento binário, $p = 0,9$
Seleção	Torneio Binário	–
Energia cinética inicial	–	10.000
Tipo de colisão	–	$0,5$
Taxa de perda de energia cinética	–	$0,2$
Limiar de síntese	–	$0,0$
Limiar de decomposição	–	$0,0$

Fonte: Próprio autor.

4.5.1 Hipervolume

O Hipervolume computa a área do polígono formado pela fronteira em relação à um ponto de referência. Essa área quantiza a qualidade da fronteira mediante à sua diversidade e convergência. Um exemplo está ilustrado na figura 12, onde a área está colorida de cinza. Ela é formada pela junção dos três pontos da fronteira com o ponto de referência. No exemplo, a fronteira escolhida é típica de um problema de minimização. Assim, o valor do hipervolume é maior toda vez que os pontos se aproximam da origem dos eixos.

Figura 12 – Hipervolume de uma fronteira de duas dimensões. O ponto de referência é o ponto r e os pontos da fronteira são $p^{(1)}, p^{(2)}, p^{(3)}$.



Fonte: Adaptado de (FONSECA; PAQUETE; LÓPEZ-IBÁÑEZ, 2006)

Para validar estatisticamente os resultados computados pelo hipervolume, foi utilizado o teste de Kruskal-Wallis. Ele tem como uso principal a verificação de que, dadas três ou mais populações, se existem diferença estatística entre as mesmas. O teste foi realizado a um nível de significância de $0,05$.

4.6 Ambiente utilizado

As devidas implementações aconteceram dentro do *framework*[1] construído para o trabalho realizado em Mariani et al. (2016). Este, por sua vez, é baseado em outro *framework*, o jMetal[2], escrito na linguagem de programação Java, o qual possui implementados diversos operadores como o de mutação (*Polynomial Mutation*) e o gerador de número aleatórios (*Mersenne Twister Generator*). Dessa forma, foi possível adotar um padrão de desenvolvimento onde cada funcionalidade desempenhou sua atividade de forma coesa.

Em relação a hiper-heurística, o *framework* utilizado no trabalho de Mariani et al. (2016) possibilitou, também, a coesão entre as funcionalidades. A generalização do código permitiu a construção da gramática utilizada neste presente trabalho bem como a implementação do SMPSO e do SPSP sem que fossem necessárias adaptações difíceis de serem feitas, contribuindo com o progresso do tema. Os experimentos foram executados em um *cluster* equipado com 12 processadores Intel Xeon Silver 4108 e 16GB de memória RAM.

[1] Disponível em: <https://github.com/GiovaniGuizzo/jMetalGrammaticalEvolution>
[2] Disponível em: <http://jmetal.sourceforge.net>

5 Resultados

Esse capítulo apresenta as comparações entre os resultados já existentes para o SPSP com as meta-heurísticas NCRO e NSGA-II e todos os SMPSOs gerados pela hiper-heurística deste trabalho. Para validar os resultados de forma estatística, cada algoritmo foi executado de forma independentemente um total de 30 vezes. Ao final, as 30 fronteiras geradas foram juntadas, e os pontos dominados ou repetidos foram excluídos. Esse procedimento permite que seja possível analisar detalhadamente o desempenho de uma meta-heurística na instância escolhida do SPSP. Para este trabalho, apenas o NCRO e o NSGA-II não foram submetidos a esse experimento, uma vez que o mesmo já tinha sido realizado da mesma forma em trabalhos anteriores.

5.1 Parametrização dos SMPSOs Gerados

A tabela 5 detalha a parametrizagem de cada um dos 9 SMPSOs gerados pela hiper-heurística. Cada meta-heurística está denominada de ALG_X, onde X varia entre $[0, 8]$, totalizando os 9 SMPSOs gerados.

Tabela 5 – Parametrização dos SMPSOs gerados pela hiper-heurística.

(a)

Parâmetro	ALG_0	ALG_1	ALG_2	ALG_3	ALG_4
População	125	125	100	100	125
Iterações	200	200	250	250	200
Mutação	Polinomial	Aleatória Simples	Polinomial	Uniforme	Polinomial
Arquivo	Grid Adaptativo	Grid Adaptativo	Grid Adaptativo	Grid Adaptativo	Grid Adaptativo
Tamanho do arquivo	125	150	120	120	150
Inicialização	Aleatória	Aleatória	Diversificada	Aleatória	Aleatória

(b)

Parâmetro	ALG_5	ALG_6	ALG_7	ALG_8
População	125	100	125	100
Iterações	200	250	200	250
Mutação	Polinomial	Polinomial	Polinomial	Polinomial
Arquivo	Grid Adaptativo	Grid Adaptativo	Grid Adaptativo	Crowding
Tamanho do arquivo	150	100	120	125
Inicialização	Aleatória	Aleatória	Diversificada	Aleatória

5.2 Análise da Fronteira

O gráfico apresentado na figura 13 compila todos os pontos dominados gerados de acordo com a estratégia descrita na seção 4.4. É possível notar que um SMPSO gerado, o ALG_8, foi o SMPSO que mais apresentou resultados acima dos computados pelo NCRO e NSGA-II. Vale lembrar que, em trabalhos anteriores realizados pelo autor, foi constatado a habilidade superior do NCRO, em particular, nessa instância escolhida do SPSP. No geral, todos os SMPSOs gerados conseguiram dominar um ou mais pontos pertencentes à fronteira do NCRO e do NSGA-II.

O ALG_8, especificamente, foi o único algoritmo que conseguiu encontrar soluções viáveis no intervalo com o custo $[1,58 \times 10^6; 1,61 \times 10^6]$. Uma possível explicação está na escolha do arquivo esclhido, este baseado na distância de *Crowding* (DEB et al., 2002). Este tipo de arquivo é um dos mais utilizados na literatura. Isto acontece, pois, uma das suas principais características é a habilidade de diversificar com mais facilidade as soluções ao longo do espaço de busca.

Dentre todos os SMPSOs gerados, com exceção do ALG_8, o desempenho foi similar. Essa característica é também observada nas tabelas 5a e 5b, onde alguns dos parâmetros não variaram muito. Por exemplo, só um algoritmo não utilizou o arquivo com o tipo Grid Adaptativo, e os tamanhos da população variou entre 100 e 125, contribuindo para a geração de uma fronteira mais similiar. Em destaque, o ALG_6 conseguiu obter bons resultados entre eles quando o custo do projeto variou no intervalo $[1,62 \times 10^6; 1,68 \times 10^6]$. Ainda, é possível perceber que os SMPSOs gerados que utilizam a mutação polinomial com uma baixa probabilidade de mutação tendem a obter melhores resultados.

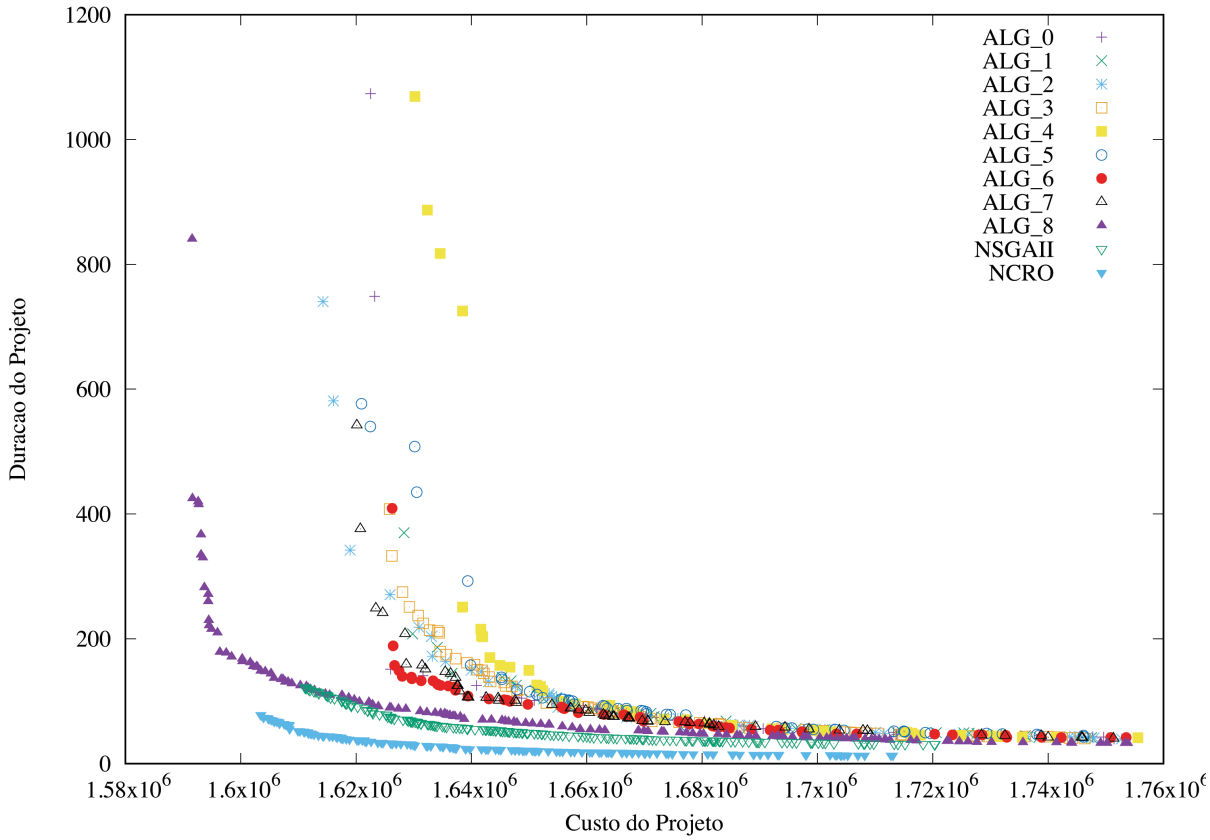
5.3 Análise do hipervolume

É comum que os resultados computados pelo hipervolume reflitam o que foi visto no gráfico da figura 13. Já que este tem seu valor calculado levando em consideração a diversidade e a convergência da fronteira, tem como consequência que fronteiras menores ou não esparsas, por exemplo, acabem tendo um valor final menor. A tabela 6 mostra o valor médio e desvio-padrão do hipervolume para cada algoritmo na instância do SPSP executada. Os valores estão no intervalo $[0; 1]$ pois todos os pontos foram normalizados segundo a fórmula 5.1. As variáveis f_i^{min} e f_i^{max} referem-se, respectivamente, ao menor e maior valor encontrados por todas as meta-heurísticas no objetivo i. Assim, o ponto de referência escolhido foi o ponto $(1, 1)$.

$$f(x)_i = \frac{(f(x)_i - f_i^{min})}{f_i^{max} - f_i^{min}} \tag{5.1}$$

Pela tabela 6, conclui-se que o SMPSO ALG_8 foi o melhor algoritmo dentre todos os outros executados, seguido pelo NCRO e NSGA-II. O teste de Kruskal-Wallis relatou diferença

Figura 13 – Gráfico de todos os pontos não dominados encontrado por cada meta-heurística.



Fonte: próprio autor.

estatísticas entre todos os resultados gerados; ou seja, os algoritmos geraram fronteiras estatisticamentes diferentes. Dessa forma, é verificada a importância que um tipo de arquivo faz no resultado final de um algoritmo. O arquivo de Crowding se mostrou capaz de manter soluções variadas, onde estas conseguiram minimizar um objetivo específico de cada vez. Essa característica também ressalta a importância de uma fronteira esparsa. Ao tomarmos o NCRO como exemplo, a sua fronteira foi a que mais conseguiu minimizar os objetivos de duração do projeto. Entretanto, por não conseguir ter uma fronteira melhor distribuída ao longo do espaço, a área do polígono acabou ficando deteriorada. Como o ALG_8 conseguiu fazer uma distribuição melhor, acabou obtendo um hipervolume maior.

Finalmente, a análise dos resultados sugere que a hiper-heurística, com a gramática definida na seção 4.3 (página 30), foi capaz de apresentar resultados competitivos diantes de resultados já encontrados na literatura, respondendo de forma **positiva** a hipótese levantada na seção 1.1. Novamente, fica comprovada a sensibilidade que meta-heurísticas possuem em relação aos seus parâmetros. Contudo, ressalvas devem ser feitas acerca dos experimentos: (1) em um contexto de urgência ou poucos recursos, o engenheiro de software poderia optar por somente uma execução da hiper-heurística, onde eventualmente um resultado ruim (como o do ALG_4) poderia ser gerado; (2) cada execução da hiper-heurística demorou cerca de 25

Tabela 6 – Valor computado do hipervolume para cada algoritmo

(a)

Algoritmo	NCRO	NSGAII	ALG_0	ALG_1	ALG_2	ALG_3
Média, Desvio	$0,885,\ 0,02$	$0,8169,\ 0,029$	$0,677,\ 0,056$	$0,6801,\ 0,050$	$0,691,\ 0,047$	$0,679,\ 0,05$

(b)

Algoritmo	ALG_4	ALG_5	ALG_6	ALG_7	ALG_8
Média, Desvio	$0,649,\ 0,05$	$0,660,\ 0,058$	$0,703,\ 0,047$	$0,683,\ 0,054$	$0,922,\ 0,02$

horas ininterruptas até a sua finalização, ainda que em um computador com bastante recursos de processamento. Sem um planejamento prévio, ou alguma otimização no algoritmo, o uso da hiper-heurística pode torna-se inviável.

6 Conclusões

Este projeto verificou a aplicação de hiper-heurísticas utilizando o método de evolução gramatical no SPSP. O tema foi estendido uma vez que o autor já havia explorado a eficiência do SMPSO para o SPSP em trabalhos anteriores. Dessa forma, até onde se sabe, este é o primeiro trabalho que aborda o método de evolução gramátical utilizando o SMPSO como meta-heurística ou o SPSP como tema. O método de evolução gramátical é uma das abordagens utilizadas por hiper-heurísticas para a geração de meta-heurísticas de forma automatizada.

Problemas como o SPSP são comumente estudados na literatura utilizando a SBSE, uma metodologia que utiliza meta-heurísticas e métricas de engenharia de software para gerar soluções de problemas nos mais diversos campos de atuação. As meta-heurísticas são algoritmos genéricos que atuam em cima de heurísticas para a geração de resultados, não garantindo, portanto, soluções ótimas para o problema. Seu resultado final depende dos seus parâmetros, que definem desde operadores à condições de parada de execução.

O cerne de um algoritmo de evolução gramátical é uma gramática, geralmente na forma BNF, onde as produções são os diferentes atributos de parâmetros de meta-heurística, e as variáveis são os possíveis valores. Dessa forma, a hiper-heurística realiza uma busca evolucionária na tentativa de achar uma parametrização que maximize os resultados de acordo com alguma métrica escolhida. A principal motivação para o uso de tais métodos é de que a escolha manual de parâmetros de execução de uma meta-heurística é feito, em sua maior parte, de forma empírica, sendo suscetível a erros ou a escolhas não ótimas de valores.

Este trabalho selecionou uma instância do SPSP, onde um possível real cenário é modelado, para que a hiper-heurística encontrasse um SMPSO que maximizasse a métrica hipervolume, a mais utilizada na literatura. Como comparação dos resultados, foram usadas as fronteiras encontradas por dois algoritmos feitos em testes anteriores, o NCRO e o NSGA-II. A análise da fronteira permitiu identificar padrões nas execuções, como a similaridade entre os SMPSOs gerados, e o hipervolume constatou que um dos algoritmos gerados sobressaiu resultados já existentes na literatura.

Há abordagens a serem investigadas no futuro. Uma delas é fazer com que a hiper-heurística seja mais eficiente no tocante à complexidade computacional, a fim de minimizar o consumo de memória e uso do processador. Nesse quesito, cabe um design e uma arquitetura de software mais robusta a ser definida. Por fim, cabe o incremento da gramática utilizada para que esta apresente mais valores de parâmetros e torne a busca mais diversificada. Na literatura, há diversos arquivos e operadores evolucionários que apresentam resultados competitivos e que podem ser aplicado ao contexto do tema deste trabalho.

Referências

ALBA, E.; CHICANO, F. Software project management with gas. *Information Sciences*, Elsevier, v. 177, p. 2380–2401, 2007. Citado 5 vezes nas páginas 8, 13, 17, 27 e 29.

ANDRADE, J. J. N.; SILVA, L.; BRITTO, A. *Uma Abordagem Para o Problema do Escalonamento Dinâmico em Projeto de Software*. Dissertação (Graduação em Ciência da Computação) — Universidade Federal de Sergipe, São Cristóvão, Abril 2018. Citado na página 12.

ANDRADE, J. J. N.; SILVA, L.; CARVALHO, A. B. Explorando o problema de escalonamento de projeto com meta-heurísticas. In: *Anais do Workshop de Trabalhos de Iniciação Científica e Graduação (WTICG)*. [S.l.: s.n.], 2017. p. 143–152. Citado na página 31.

BECHIKH, S.; CHAABANI, A.; SAID, L. B. An efficient chemical reaction optimization algorithm for multiobjective optimization. *IEEE Transactions on Evolutionary Computation*, IEEE, v. 45, n. 10, p. 2051–2064, 2015. Citado na página 31.

BEUME, N. et al. On the complexity of computing the hypervolume indicator. *IEEE Transactions on Evolutionary Computation*, IEEE, v. 13, n. 5, p. 1075–1082, 2009. Citado na página 31.

BRUCKERA, P. et al. Resource-constrained project scheduling: Notation, classification, models, and methods. *European Journal of Operational Research*, v. 112, n. 1, p. 3–41, January 1999. Citado na página 8.

BURKE, E. K. et al. Hyper-heuristics: A survey of the state of the art. *Journal of the Operational Research Society*, Springer, v. 64, n. 12, p. 1695–1724, 2013. Citado 3 vezes nas páginas 9, 24 e 25.

CHANG, K. C. et al. Time-line based model for software project scheduling with genetic algorithms. Elsevier, v. 50, p. 1142–1154, 2008. Citado na página 8.

CHEN, W.; ZHANG, J. Ant colony optimization for software project scheduling and staffing with an event-based scheduler. IEEE, v. 39, p. 1–17, 2013. Citado na página 8.

CHEN, W.; ZHANG, J. Ant colony optimization for software project scheduling and staffing with an event-based scheduler. *IEEE Transactions on Software Engineering*, IEEE, v. 39, p. 1–17, March 2013. Citado na página 13.

COELLO, C. A. C.; LAMONT, G. B.; VELDHUIZEN, D. A. V. *Evolutionary Algorithms for Solving Multi-Objective Problems*. 2nd. ed. [S.l.]: Springer, 2007. (Genetic and Evolutionary Computation). Citado 3 vezes nas páginas 11, 17 e 18.

CRAWFORD, B. et al. A max–min ant system algorithm to solve the software project scheduling problem. *Expert Systems with Applications*, Elsevier, v. 41, n. 15, p. 6634–6645, 2014. Citado 2 vezes nas páginas 8 e 17.

DEB, K. et al. A fast and elitist multiobjective genetic algorithm: Nsga-ii. *IEEE Transactions on Evolutionary Computation*, IEEE, v. 6, n. 2, p. 182–197, Agosto 2002. Citado 2 vezes nas páginas 31 e 35.

FERRUCCI, F.; HARMAN, M.; SARRO, F. Search-based software project management. In: RUHE, G.; WOHLIN, C. (Ed.). *Software Project Management in a Changing World*. [S.l.]: Springer Berlin Heidelberg, 2014. p. 373–399. ISBN 978-3-642-55034-8. Citado na página 13.

FONSECA, C. M.; PAQUETE, L.; LÓPEZ-IBÁÑEZ, M. An improved dimension-sweep algorithm for the hypervolume indicator. In: *Proceedings of the 2006 Congress on Evolutionary Computation (CEC 2006)*. Piscataway, NJ: IEEE Press, 2006. p. 1157–1163. Citado na página 32.

GAREY, M. R.; JOHNSON, D. S. *Computers and Intractability: a guide to the theory of np-completeness*. 1th. ed. [S.l.]: W. H. Freeman, 1979. (Series of Books in the Mathematical Sciences). Citado na página 17.

GOLDBARG, M. R.; GOLDBARG, E. G.; LUNA, H. P. L. *Otimização combinatória e meta-heurísticas: algoritmos e aplicações*. 1th. ed. Rio de Janeiro: Elsevier, 2016. Citado 4 vezes nas páginas 8, 11, 17 e 18.

GROUP, T. S. *CHAOS*. [S.l.], 2015. Citado 2 vezes nas páginas 8 e 13.

GUIZZO, G. et al. A hyper-heuristic for the multi-objective integration and test order problem. In: ACM. *Proceedings of the 2015 Annual Conference on Genetic and Evolutionary Computation*. [S.l.], 2015. p. 1343–1350. Citado na página 9.

HARMAN, M. The current state and future of search based software engineering. In: *Future of Software Engineering*. DC: IEE Computer Society, 2007. p. 342–357. Citado na página 9.

HARMAN, M. Search-based software engineering: Trends, techniques and applications. *ACM Computing Surveys (CSUR)*, v. 45, n. 1, November 2012. Citado 2 vezes nas páginas 16 e 18.

HARMAN, M.; CLARK, J. Metrics are fitness functions too. In: *10th International Symposium on Software Metrics*. Chicago, Illinois, EUA: IEEE, 2004. p. 58–69. Citado na página 17.

HARMAN, M.; JONES, B. F. Search-based software engineering. *Applied Soft Computing*, Elsevier, v. 43, n. 14, p. 833–839, 2001. Citado 2 vezes nas páginas 8 e 16.

JIA, Y. Hyperheuristic search for sbst. In: *Search-Based Software Testing (SBST), 2015 IEEE/ACM 8th International Workshop on Search-Based Software Testing*. Italy: IEEE, 2015. Citado na página 9.

JIA, Y. et al. Learning combinatorial interaction test generation strategies using hyperheuristic search. In: IEEE. *Software Engineering (ICSE), 2015 IEEE/ACM 37th IEEE International Conference on*. [S.l.]: IEEE, 2015. v. 1, p. 540–550. Citado na página 9.

KENNEDY, J.; EBERHART, R. Particle swarm optimization. In: *Neural Networks, 1995. Proceedings., IEEE International Conference on Neural Networks*. [S.l.: s.n.], 1995. v. 6. Citado na página 19.

KITCHENHAM, B. *Procedures for Performing Systematic Reviews*. [S.l.], 2004. Citado na página 27.

KITCHENHAM, B.; CHARTERS, S. *Guidelines for performing Systematic Literature Reviews in Software Engineering*. [S.l.], 2007. Citado na página 27.

KOZA, J. *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. 1st. ed. Massachusetts: A Bradford Book, 1992. (Complex Adaptive Systems). Citado na página 21.

LAM, A. Y. S.; LI, V. O. K. Chemical-reaction-inspired metaheuristic for optimization. *IEEE Transactions on Evolutionary Computation*, IEEE, v. 14, n. 3, p. 381–399, 2010. Citado na página 18.

LUNA, F. et al. The software project scheduling problem: A scability analysis of multi-objective metaheuristics. *Applied Soft Computing*, v. 15, p. 136–148, 2013. Citado na página 11.

MARIANI, T. *An Approach Based on Hyper-Heuristic To Generate Algorithms For Refactoring Object-Oriented Programs*. Tese (Doutorado) — Federal University of Paraná, Curitiba, 2017. Citado 3 vezes nas páginas 9, 23 e 24.

MARIANI, T. et al. Grammatical evolution for the multi-objective integration and test order problem. In: *Proceedings of the Genetic and Evolutionary Computation Conference 2016*. [S.l.: s.n.], 2016. p. 1069–1076. Citado 2 vezes nas páginas 9 e 33.

NAUR, P. et al. Revised report on the algorithmic language algol 60. In: *ALGOL-like Languages*. [S.l.]: Springer, 1997. p. 19–49. Citado na página 22.

NEBRO, A. J. et al. Smpso: A new pso-based metaheuristic for multi-objective optimization. In: IEEE. *Computational intelligence in miulti-criteria decision-making, 2009. mcdm'09. ieee symposium on*. [S.l.], 2009. p. 66–73. Citado na página 20.

O'NEILL, M.; RYAN, C. Grammatical evolution. *IEEE Transactions on Evolutionary Computation*, v. 5, n. 4, p. 349–358, August 2001. Citado na página 22.

PEIXOTO, D. C. C.; MATEUS, G. R.; RESENDE, R. F. *Evaluation on the Search-Based Optimization Techniques to Schedule and Staff Software Projects: a Systematic Literature Review*. 2014. Disponível em: <http://homepages.dcc.ufmg.br/~cascini/cascini_paper_SBSE.pdf>. Citado 2 vezes nas páginas 8 e 13.

PRESSMAN, R. S. *Software Engineering: A Practitioner's Approach*. 8th. ed. Nova Iorque: McGraw-Hill Education, 2014. Citado 2 vezes nas páginas 8 e 13.

REYES-SIERRA, M.; COELLO, C. C. Multi-objective particle swarm optimizers: A survey of the state-of-the-art. *International journal of computational intelligence research*, v. 2, n. 3, p. 287–308, 2006. Citado na página 19.

SABAR, N. R. et al. Grammatical evolution hyper-heuristics for combinatorial optimization problems. *IEEE Transactions on Evolutionary Computation*, IEEE, v. 17, n. 6, p. 840–861, September 2013. Citado 2 vezes nas páginas 9 e 25.

SIPSER, M. *Introduction to the Theory of Computation*. 3rd. ed. [S.l.]: Cengage Learning, 2012. Citado na página 22.

SOMMERVILLE, I. *Software Engineering*. 10th. ed. [S.l.]: Pearson, 2015. Citado 2 vezes nas páginas 8 e 12.

WIKIPÉDIA. *Pareto efficiency — Wikipedia, The Free Encyclopedia*. 2018. <https://en.wikipedia.org/w/index.php?title=Pareto_efficiency&oldid=833551318>. Citado na página 12.

XIAO, J.; AO, X.; TANG, Y. Solving software project scheduling problems with ant colony optimization. *Computers & Operations Research*, Elsevier, v. 40, p. 33–46, 2013. Citado na página 8.